haute école
neuchâtel berne jura

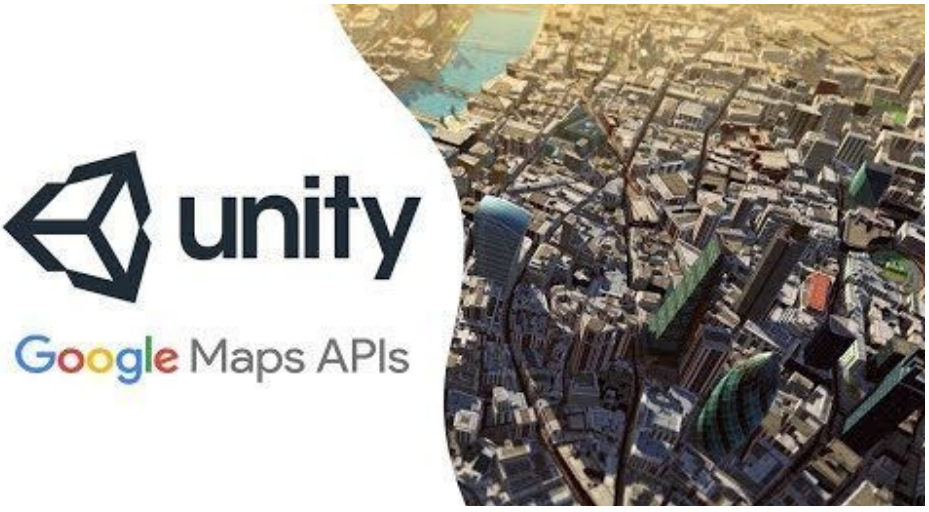
Google Maps SDK & Unity

Ce rapport est le résultat du projet P3 Google Maps dans Unity réalisé à la Haute École Arc dans la filière Informatique/Développement logiciels et multimédia.

Rapport de projet , n°206

Enseignant : Julien Senn

Etudiants : Dos Santos Ferreira Julien



Date : 29 janvier 2021

Version : 1.0.0

Table des matières

Liste des tableaux

Table des figures

1 Abstract

Ce rapport est le résultat du projet de troisième année réalisé dans le cadre d'études de Bachelor en développement logiciel et multimédia à la Haute école ARC de Neuchâtel.

Le projet concisite en l'étude des possibilités offertes par le SDK de Google Map à destination du moteur de jeu multiplatforme Unity.

Le projet est divisé en trois phases. La première est la prise en main et l'apprentissage du moteur de jeu Unity. La deuxième phase est une phase de recheches et de tests des possibilités du SDK. Finalement la troisième phase conciste en la réalisation d'une application Android reprennant les fonctionnalités testées afin de fournir un exemple d'utilisation du SDK dans un projet concret.

This report is the result of the third year project carried out as part of the Bachelor studies in software and multimedia development at the ARC High school in Neuchâtel.

The project is concise by studying the possibilities offered by the Google Map SDK for the Unity multi-platform game engine.

The project is divided into three phases. The first is getting started with and learning the Unity game engine. The second phase is a phase of research and testing the possibilities of the SDK. Finally the third phase consists in the realization of an Android application using the tested functionalities in order to provide an example of use of the SDK in a concrete project.

2 Introduction

Dans le cadre du projet de troisième année à la haute école arc de Neuchâtel dans la filière informatique développement logiciel et multimédia, il est demandé de réaliser un travail individuel proposé par un enseignant ou collaborateur de l'école.

Ce rapport est le résultat du projet proposé par M. Senn et réalisé par M. Dos Santos Ferreira. Le projet consiste en la recherche des possibilités et fonctionnalités offertes par le SDK de Google Maps à déstination de Unity.

Le projet est séparé en trois phases :

- La première phase est une phase de formation pour l'étudiant. Cette phase conciste en la prise en main et l'apprentissage de base de l'outil de développement Unity 3D.
- La deuxième phase est une phase de recherches, de tests et de documentation des possibilités offertes par le SDK.
- La troisième phase correspond à la réalisation d'une application mobile à destination de la platforme Android. L'application regroupe certaines fonctionnalités étudiée lors de la première phase afin d'avoir une synthèse de ce qu'il est possible de réaliser comme projet avec le SDK.

2.1 Composition du rapport

Ce rapport décrit le déroulement du projet en commençant par sa conception. Il contient également l'entier de la documentation réalisé lors de la phase de recherche et finalement la documentation concernant la réalisation de l'application mobile.

2.2 Le SDK de Google Maps pour Unity

Lors de la Google I/O ’18 (Séminaire de présentation des nouveau produits / services de Google de 2018) Google annonce l’arrivé d’un SDK de Google Maps à destination des développeurs Unity. Le SDK est principalement présenté sous la forme d’un outil permettant au développeur de jeux vidéo utilisant le moteur Unity 3D de réaliser des jeux dont l’environnement dans lequel évolue le joueur correspond au monde réel. Ce monde offert par les services de Google Maps comprend des modèles 3D pour chaque bâtiments du monde en plus des modèle 3D bien plus détaillés de certains monuments connus tel que la tour Eiffel, la statue de la liberté, etc. Le monde entier est disponible à l’exception de certains pays tel que la Corée du Nord.

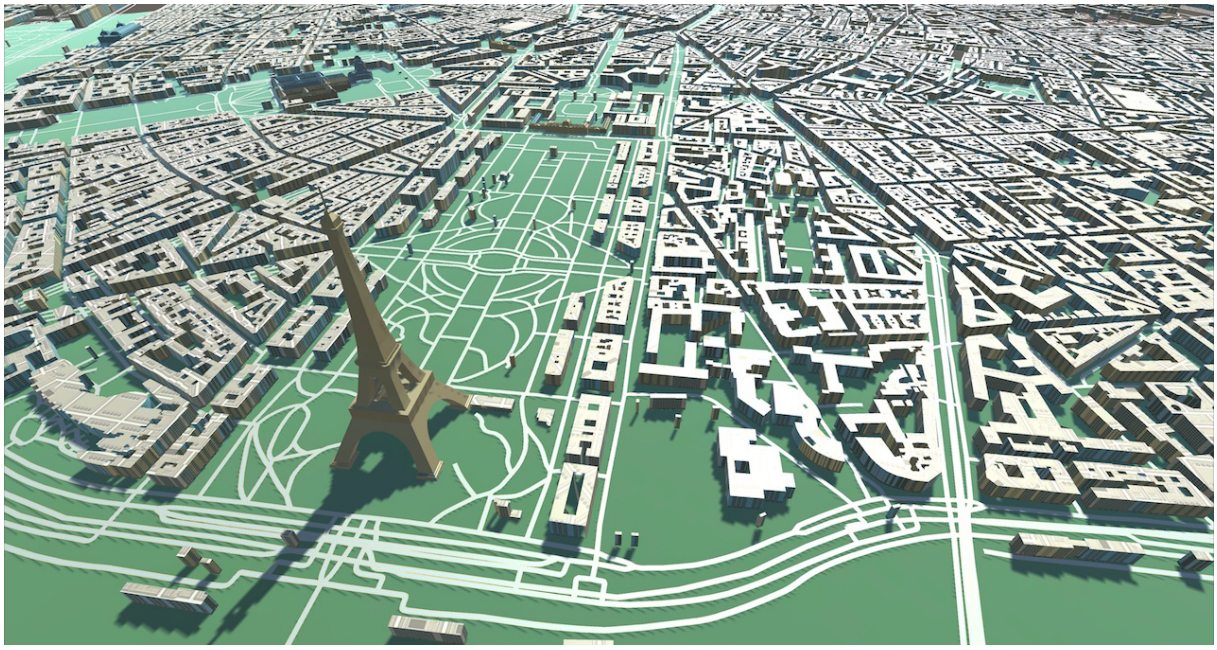


Figure 1 – Image de présentation du SDK

(Source : https://developers.google.com/maps/documentation/gaming/overview_musk)

3 Spécialisation

3.1 Cahier des charges

Le cahier des charges du projet à été défini au départ du projet conjointement entre l'étudiant et le superviseur. Dans sa première version validé par les deux partis, le cahier des charges détaillais une application final mettant en place une base de gameplay pouvant être réutilisé dans le cadre du projet dans le cours d'infographie.

Il s'est avéré que dans le cadre du projet d'infographie l'idée de continuer sur la base d'application développer dans ce projet n'a pas été retenu. Après discussion entre le superviseur et l'étudiant, il à été décidé de changer la description d'application pour l'orienter vers une application qui se suffit à elle même pour démontrer ce qu'il est possible de faire avec le SDK de Google Map.

Par conséquent le cahier des charges décrit ci-dessous correspond à la version modifié en cours de route mais validé par les deux partis à nouveau.

3.1.1 Situation initiale

Ce projet se déroule dans le cadre du projet de troisième année à la Haute École Arc dans la filière informatique, développement logiciel et multimédia. M. Senn propose un projet de recherche autour du SDK pour Unity offert par Google pour son service Maps.

3.1.2 Buts du projet

Le but du projet est de prendre en mains Unity et le SDK de Google Maps afin de déterminer les capacités et limitations de ce dernier. Le projet se découpe en trois grandes phases :Prise en mains de UnityRecherche des capacités/limitations du SDK Google Maps dans UnityRéalisation d'une application Android utilisant les possibilités du SDK Au terme de la deuxième phase du projet, chaque fonctionnalité étudiée s'accompagnera d'une documentation et d'un petit programme permettant de tester cette dernière. La troisième phase a pour but de réaliser une application comprenant plusieurs des fonctionnalités étudiées lors de la deuxième phase. Ce logiciel sera une application Android tel que Pokémon Go, Minecraft Earth ou encore Harry Potter Wizard Unite. A la différence de ces derniers le joueur n'évoluera pas sur une simple carte du monde réel mais au milieu des bâtiments en 3D. Le jeu sera soit jouable en se déplaçant dans le monde réel ou en utilisant un joystick sur l'écran. Le joueur aura pour but de ramasser des ressources plus ou moins rares en fonction de leurs emplacements. Le but de cette application est de mettre en place une base de gameplay qui pourra être utilisée dans le cadre de la réalisation du projet d'infographie avec M. Le Callennec.

3.1.3 Mise à jour des buts du projet (17.11.2020)

Le projet ne change pas drastiquement, mais une mise à jour de l'objectif final a été discuté avec l'enseignant. La phase de tests ne change absolument pas, il est toujours nécessaire de réaliser un travail de recherche et de test afin de déterminer les possibilités offertes par le SDK de Google Maps pour Unity. En revanche l'objectif d'application final est un peu différent. Dans le cours d'infographie l'idée de réaliser la suite du jeu proposé n'est plus d'actualité. Ceci entraîne le désir de changement d'application par les deux partis (enseignant et étudiant) dans le but de fournir une application «qui se suffit à elle-même» pour démontrer un exemple d'utilisation du SDK dans un projet. Le nouveau projet consiste donc en la visite de la ville de Neuchâtel en 3D. L'utilisateur a la possibilité de se déplacer dans la ville de points d'intérêts en points d'intérêts afin de découvrir cette dernière. Il a également la possibilité d'interagir avec ces points d'intérêts afin d'en afficher une description plus complète. Afin de gameifier la visite, sur son chemin, l'utilisateur a pour but de récolter des ressources afin d'augmenter son score. L'utilisateur doit être guidé, au moins savoir où se trouve les points d'intérêts.

3.1.4 Objectifs principaux

- Prise en main de Unity et du SDK Google Maps pour Unity
- Recherche des possibilités offertes par le SDK
- Documentation et réalisation de logiciel test pour chaque fonctionnalité
- Réalisation de la base de l'application
  - Monde (2D, carte)
  - Implémentation du joueur dans le monde
  - Déplacements (Géolocalisation ou joystick)
  - Affichage d'éléments à des coordonnées précises provenant de la base de données
  - Interactions avec les éléments provenant de la base de données (Affichage de la description d'un bâtiment tel que « Atelier de transformation » ou encore « Magasin »).

3.1.5 Objectifs secondaires

- Page de paramètre permettant de sélectionner le mode de déplacement
- Génération aléatoire de petits éléments autour du joueur (Ressources)
- Récolte des ressources
- Sauvegarde locale de la partie (Nombre de ressources, paramètres, etc.)
- Ajout des bâtiments 3Ds (Fonctionnalité du SDK)

3.2 Organisation du projet

3.2.1 Délais

- Début du travail : lundi 14 septembre 2020
- Remise des délivrables : jusqu'au vendredi 29 janvier 2021 à 17h00
- Défense du projet : mardi 2 février 2021 selon planning

3.2.2 Participants

- Etudiant : Dos Santos Ferreira Julien
- Superviseur : Senn Julien

3.2.3 Type de projet

Le projet est interne à l'école

4 Conception

4.1 Planification

La planification se trouve également en annexe de ce rapport.

Lors du déroulement, la planification réalisée au départ du projet et se trouvant en annexe `Gantt_of_project.xslx` n'a pas été suivie à la lettre. La phase de tests du projet a pris beaucoup plus de temps que prévu mais finalement le retard accumulé durant cette phase s'est vite rattrapé lors de la phase de réalisation de l'application car cette dernière est en partie une composition de chaque tests réalisés.

Ci-dessous vous pouvez voir le diagramme de Gantt. Chaque bloc de tâches sur la gauche correspond à une des phases du projet.

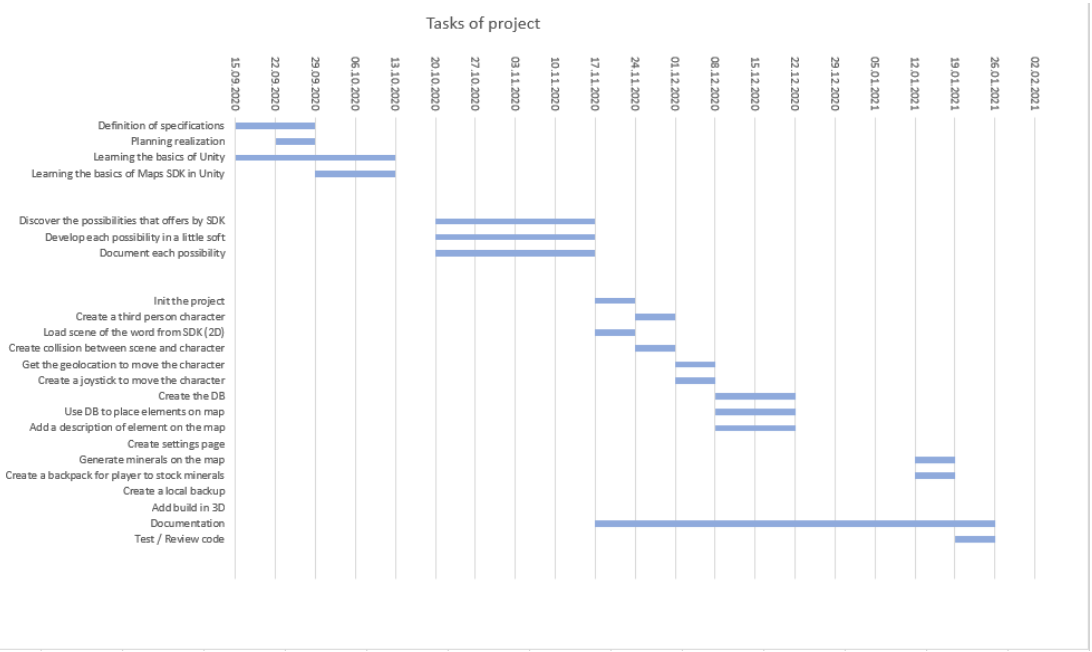


FIGURE 2 – Plannification

5 Réalisation

Le projet étant découpé en trois phases, ce chapitre se retrouve découpé en trois grandes parties directement liés aux phases du projet.

5.1 Workflow Git

Lors du développement du projet la forge de l'école, GitLab, à été utilisée. Sur cette forge le worflow git utilisé est le workflow de branche par fonctionnalités.

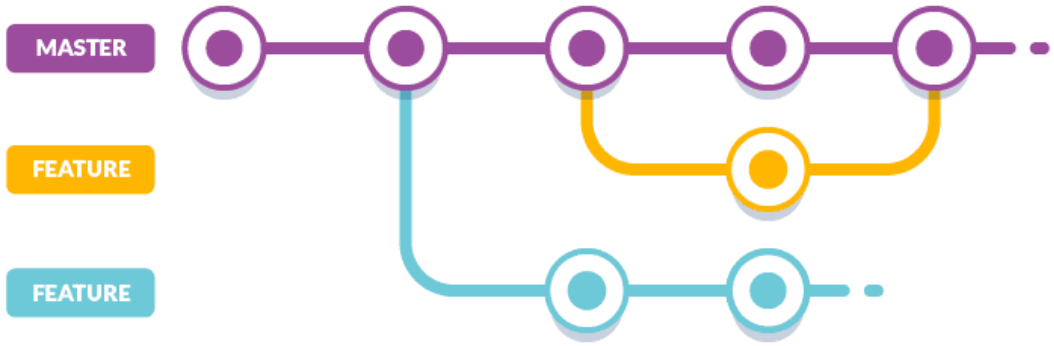


FIGURE 3 – Workflow Git

(Source : https://buddy.works/blog/5-types-of-git-workflows)

5.2 L'environnement de développement

TABLE 1 – Logiciels et versions

Logiciel	Version
OS	Ubuntu 20.04 (LTS)
Unity	2019.4.10f1 (LTS)
PHP	7.4.3
MySQL	15.1 Distribution 10.3.25-MariaDB
Visual Studio Code	1.52.1
Postman	7.36.1
Git	2.25.1
Python	3.8.5

L'entier du projet à été réalisé sur Linux avec la distribution Ubuntu 20.4. La version Linux de Unity n'est pas des plus stable. Il arrive souvent que le logiciel crash et ne réponde plus. Pour quelqu'un qui commencerait un projet avec Unity il peut être préférable de le réaliser sur Windows.

5.3 Phase 1 L'apprentissage de Unity

Dans cette première phase plusieurs tutoriels sur Unity 3D on été réalisés.

Dans un premiers temps des tutoriels simple de prise en main de environnement de développement on été réalisés.

Puis un tutoriel permettant de réaliser un contrôleur de jeu en vue à la troisième personne à été réalisé. Il permet de prendre dans l'avance dans la conception des scènes de test lié au fonctionnalités découvertes et testé. Il permet de se déplacer dans les scènes.

5.3.1 La représentation du joueur

Un cylindre rouge représente le corps du joueur et un rectangle placé au dessus du cylindre représente la tête du joueur et permet de savoir dans quel direction ce dernier regarde. Cette composition est placé dans un GameObject parent appelé `ThirdPersonPlayer`

Le GameObject parent contient un composant `Rigidbody` et le corps contient un composant `Capsule Collider` un peu plus large que l'ensemble du joueur. Ces derniers composants serviront à la collision entre le joueur et les bâtiments.

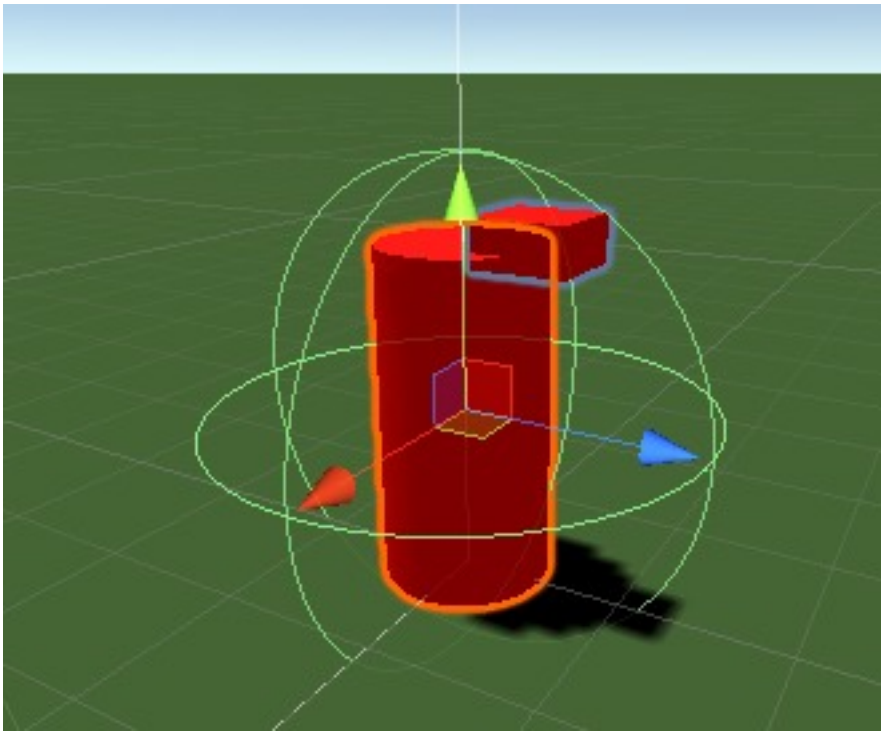


Figure 4 – Composition du joueur

5.3.2 Cinemachine

Cinemachine est un package permettant de définir des comportements de camera adapté au contrôleurs de jeu tel qu'un contrôleur de jeu en vue à la troisième personne.

Un deuxième GameObject `ThirdPersonCamera` contient les composants `CinemachineFreeLook` et `Cinemachine Collider`.

Une fois ce GameObject créé, il est possible de configuré la cible à suivre en permanence avec la camera dans les propriétés du composant `CinemachineFreeLook`. Dans ce cas, le joueur.

Figure 5 – Paramètres de Cinemachine

De plus, le comportement se base sur la navigation entre trois cercles pouvant être paramètre également.

Le composant `Cinemachine Collider`permet de gérer automatiquement les collisions entre la camera et les autres objets de la scènes.

La camera utilisée par cinemachine est la camera principale avec l'ajout d'un composant `CinemachineBrain`

5.4 Phase 2 La recherche et le test de fonctionnalités

Les différents chapitres ci-dessous documentent les fonctionnalités principales et utiles au développement de l'application finale du projet.

5.4.1 Les exemples

Ce chapitre est directement lié au logiciel d'exemple se trouvant dans

```
1 01_Tests/soft_example_tests
```

Chaque chapitres documentes un test d'une possibilité du SDK et est un complément à une scène du projet Unity cité au-dessus. Ces scènes permettent d'avoir un exemple d'implémentation de la fonctionnalité testé et documenté dans ce rapport.

5.4.2 Chargement d'une zone du monde dans la scène

Ce chapitre décrit l'import du SDK de Google Maps pour Unity dans un projet Unity et l'utilisation de ce dernier dans le but de charger une zone géographique du monde dans une scène du projet grâce à une latitude et une longitude.

Dans la scène de test vous retrouverez un contrôleur en vue à la troisième personne afin d'explorer la zone importée.

Prérequis

- Afin d'utiliser le SDK fournis par Google, il faut utiliser une version LTS de Unity (2018 ou 2019).
- Obtenir une clé pour L'API Semantic Tile de Google*

*La facturation doit être activé sur le compte qui génère la clé d'API.

Importer le SDK Dans un premier temps, il faut créer un projet Unity 3D.

Ensuite téléchargez le SDK a cette adresse :

- 2019 : https://www.gstatic.com/mapsunitysdk/maps_sdk_for_unity_2019.zip
- 2018 : https://www.gstatic.com/mapsunitysdk/maps_sdk_for_unity_2018.zip

Une fois le SDK télécharger, il faut l'importer dans notre projet. Pour ce faire :

- Assets -> Import Packages -> Custom Package

Sélectionnez ensuite : All puis Import

Première utilisation du SDK Afin d'utiliser le SDK dans notre projet, il faut créer un objet vide dans le projet, renommez le "Map Base", puis ajoutez lui le composant "Map Service".

```
C'est ce composant qui fera le lien avec le SDK.
```

Dans le champs "API Key" collez votre clé d'API.

Ajoutez ensuite un autre composant, "Basic Exemple" à l'objet "Map Base".

```
Ce composant permet de spécifier une localisation, sous la forme de latitude
 et longitude (Avec décimal et non DMS), comprise dans la zone àcharger.
```

Il ne reste plus qu'à lancer le programme, et nous voilà dans la zone spécifiée.

Dans le soft d'exemple, vous vous retrouverez dans le quartier de la gare de Neuchâtel, proche de la HE-Arc (46.996619, 6.93683799999997).

5.4.3 Chargement automatique du monde en fonction des déplacements du contrôleur

Ce chapitre décrit le chargement automatique du monde fournis par le SDK de Google Maps dans Unity quand le contrôleur se déplace dans le monde.

Dans la scène de test vous retrouverez un contrôleur en vue à la troisième personne afin d'explorer la zone importée et de tester les chargements automatiques.

Prérequis

- Afin d'utiliser le SDK fournis par Google, il faut utiliser une version LTS de Unity (2018 ou 2019).
- Obtenir une clé pour L'API Tile de Google*
- Avoir importé le SDK dans votre projet.

*La facturation doit être activé sur le compte qui génère la clé d'API.

Les composants du SDK Dans un premier temps, créez un objet vide dans votre scène (`MapsSDK`). Ensuite ajoutez lui les composants suivant : * Map Service

```
Ce composant permet la liaison avec le SDK.
```

- Dynamic Maps Updater

```
Ce composant permet le chargement automatique du monde lorsque'il y a dé
placement de la Main Camera.
```

- Base Map Loader

```
Ce composant permet le premier chargement de la zone de départ, il permet é
galement de spécifier les coordonnées latitudes et longitudes de départ.
```

Une fois importé, il faut spécifier certains attributs à ces composants. * Map service : * Attribut API Key -> votre clé d'API

- Dynamic Maps Updater
  - Attributs Base Map Loader -> Sélectionnez l'objet de votre scène qui comprend le composant du même nom que l'attribut.
  - Attribut Ground -> Sélectionnez votre objet correspondant au sol de votre monde
  - Attribut Camera Move Distance -> Correspond à la distance parcouru par la Main Camera avant de demander le chargement automatique de la zone.
  - Attribut Camera Move Angle -> Correspond à l'angle en degrés que la Main Camera doit faire avant de demander le chargement automatique de la zone.
- Base Map Loader
  - Attribut Maps Service -> Sélectionnez l'objet de votre scène qui comprend le composant du même nom que l'attribut.

— Attribut Lat Ling -> Donnez les coordonnés de la zone de départ voulu sous la forme de Latitude, Longitude à virgule et non DMS.
— Attribut Max Distance -> Permet de spécifier la taille de la première zone à charger (Les tests ont été réalisés avec la valeur 500).

Problèmes / Solutions On peut se rendre compte que Le SDK offre cette possibilité simple de recharger la carte dynamiquement en fonction de la distance parcouru par la camera spécifiée. Cela entraîne obligatoirement que pour réaliser les déplacements du contrôleur, c'est ce dernier qui doit se mouvoir dans le monde entraînant avec lui la camera.

Il semble que Google ne se soit pas aligné sur les différents standards de la communauté de développeurs Unity car après discussion avec M. Le Callenec et quelques recherches sur internet, il semblerait que la communauté aie pour habitude de déplacer le monde qui entour le jour plutôt que le joueur lui même dans le monde lorsque le joueur risque de se déplacer loin du point d'origine de la scène. Ceci afin de ne pas perdre en précision. Le SDK propose lui plutôt de déplacer le joueur dans le monde et après une certaines distance parcourue, de réinitialiser le point d'origine de la scène sur l'emplacement actuel du joueur.

Il s'avère qu'en déplacent directement le contrôleur et par conséquent la caméra, il arrive parfois des lags au moment du rechargement de la zone et de la mise à jour du point d'origine flottante (explications à la section suivante). Ces lags proviennent du faite que pour le contrôleur l'outil `Cinemachine` est utilisé afin d'obtenir des mouvements de cameras correspondant à une vue à la troisième personne. Les lags correspondent à un reset de la camera. Pour contrer ces lags et donc obtenir un déplacement fluide sans lags même au moment du rechargement de la zone, une solution à été trouvé en discutant du problème avec M. Le Callenec. Dans unity, de manière général, on ne déplace pas le contrôleur directement mais on déplace le monde qui l'entour. Cette technique permet d'éviter des lags lors des déplacements.

Afin de mettre ceci en place, le script gérant les déplacements du contrôleur est modifié de manière à ne plus déplacer ce dernier mais le gameObject `MapsSDK` car il est le parent du monde (Bâtiments, routes, cours d'eau, etc.). A l'exécution tous les gameObjects du monde se retrouvent enfant de `MapsSDK`.

Il faut également modifier une ligne du script `DynamicMapsUpdater` fournis par le SDK afin de faire fonctionner le rechargement dynamique du monde avec notre nouveau système de déplacement. Comme expliqué plus haut, ce dernier se base sur la distance parcourue par la camera avant de recharger / décharger la zone. Dans notre solution la camera ne se déplace plus, il faut donc se baser sur les déplacements du monde.

Dans la méthode `Update()` il faut transformer la ligne 108

```
1  Vector3 cameraPosition = Camera.main.transform.position;
```

en

```
1 Vector3 cameraPosition = transform.position;
```

(Le composant du script est ajouté directement au gameObject `MapsSDK` et c'est ce dernier qui sera déplacé, donc on accède à sa position directement comme attributs.)

L'entier du script fonctionnera de la même manière que pensé par Google car même si le déplacement du monde est l'inverse du déplacement du contrôleur le calcul de la distance se fait de manière vectoriel et donc en valeur absolue.

Mise à jour du point d'origine Le SDK fournis un système de conversion des positions données par une latitude et une longitude en vecteurs dans la scène Unity. Ce système est également utilisé en interne par le SDK afin de placer le monde correctement.

Cette conversion se fait sur la base de nombre décimaux. Le problème des nombres décimaux est bien connu en informatique. Les ordinateurs ne peuvent pas stockés ou calculés des nombres décimaux avec une précision infinie. Au départ, le point donné au composant `Base Map Loader` correspond à l'origine (0,0,0) de la scène. Le point d'origine flottant (`floatingOrigin` pour le SDK) est donc setter sur ce point.

Plus le joueur se déplace loin de cette origine dans le monde, plus la conversion des positions latitude/longitude des objets du monde devient imprécise et donc l'erreur de placement dans la scène Unity devient de plus en plus grande.

Pour contrer ce problème le SDK fournis une méthode permettant de mettre à jour le `floatingOrigin`. Et donc ramener une position au centre de la scène Unity à un moment donné.

Dans notre cas, la mise à jour du point d'origine flottant se fait à chaque fois que le joueur s'est déplacé d'une valeur de 100. Le code permettant cette mise à jour se trouve dans le script de déplacements du joueur

```
1 Vector3 newOrigin = GetWorldPositionOnGroundPlane();
2 if(Vector3.Distance(floatingOrigin, newOrigin) > 100){
3     MapsService.MoveFloatingOrigin(newOrigin);
4     MapsService.transform.position = new Vector3(0,0,0);
5     floatingOrigin = newOrigin;
6 }
```

Une fois l'origine flottante redéfinie, il faut replacer le monde sur l'origine de la scène.

La méthode `GetWorldPositionOnGroundPlane()` retourne la position du monde en ignorant la valeur vertical Y et en inversant les valeurs des axes X et Z car le déplacement du monde est inverse au déplacement du joueur.

```
1 private Vector3 GetWorldPositionOnGroundPlane() {
2     Vector3 result = MapsService.transform.position;
3     result.y = 0;
4     result.x = -result.x;
5     result.z = -result.z;
6     return result;
7 }
```

Dans le mini-soft de test vous démarrez dans le quartier de la gare de Neuchâtel, vous pouvez librement vous déplacer et tester le chargement automatique du monde avec la mise à jour du point d'origine flottante.

5.4.4 Collisions entre le contrôleur et les bâtiments

Ce chapitre décrit le script développé afin de mettre en place les collisions entre le contrôleur et les bâtiments générés par l'API de Google Maps.

Dans la scène de test vous retrouverez un contrôleur en vue à la troisième personne afin d'explorer le monde et de tester les collisions.

Prérequis

- Afin d'utiliser le SDK fournis par Google il faut utiliser une version LTS de Unity (2018 ou 2019).
- Obtenir une clé pour L'API Tile de Google*
- Avoir importé le SDK dans votre projet.

*La facturation doit être activé sur le compte qui génère la clé d'API.

Les composants Afin de permettre les collisions entre le joueur et les bâtiments généré par le SDK, il faut ajouter au joueur un composant `Rigidbody` en le paramétrant comme sur l'image ci-dessous afin de bloquer la position en y et la rotation sur tous les axes.

Configuration du composant Rigidbody

Il faut également ajouter le composant `Capsule collider` sur l'objet composant le corps de joueur.

Le script Le script s'ajoute sur en composant de l'objet "MapSDK" de la scène. Il permet d'ajouter les composants `Mesh collider` à chaque bâtiments générés par le SDK.

```
1   public class CollisionScript : MonoBehaviour
2   {
3       /// <summary>
4       /// The <see cref="MapsService"/> with which to register our
            event handlers.
5       /// </summary>
6       public MapsService MapsService;
7
8       /// <summary>
9       /// The object to use as the player's avatar.
10      /// </summary>
11      public GameObject Player;
12
13      // Start is called before the first frame update
14      void Start()
15      {
16          MapsServiceSetup();
17      }
18
19      /// <summary>
20      /// Setup <see cref="MapsService"/>. Give buildings mesh
            colliders.
21      /// </summary>
22      private void MapsServiceSetup()
23      {
24          MapsService.Events.ExtrudedStructureEvents.DidCreate.
                AddListener(args =>
25          {
26              // Assign every building a mesh collider, so player can
                    collide with them.
27              MeshFilter meshFilter = args.GameObject.GetComponent<
                    MeshFilter>();
28              MeshCollider collider = args.GameObject.AddComponent<
                    MeshCollider>();
29
30              collider.sharedMesh = meshFilter.sharedMesh;
31          });
32      }
33  }
```

Le `Mesh collider` va s'aligner sur le composant `Mesh` des bâtiments afin de parfaitement coller à leurs formes.

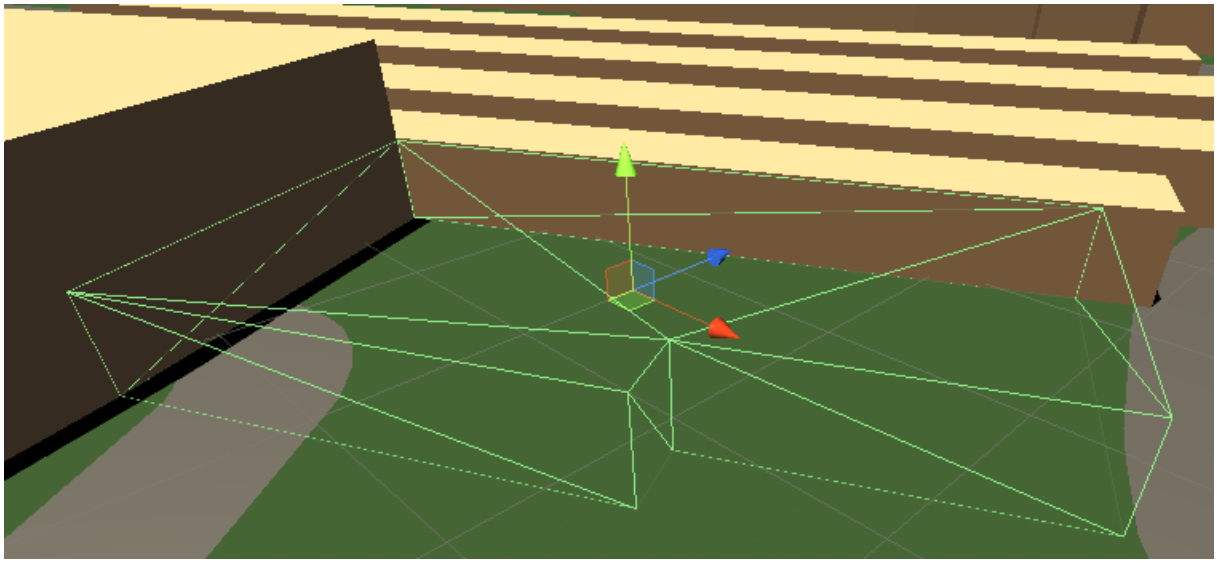


Figure 6 – Mesh Collider

5.4.5 Chargement d'objet provenant d'une base de donnée externe

Ce chapitre décrit comment mettre en place un système permettant de charger des objets provenant d'une base de données externe à l'application et au SDK et comment placé ces derniers dans le monde généré par le SDK.

Dans la scène de test vous retrouverez un contrôleur en vue à la troisième personne afin d'explorer la zone et de voir les différents points d'intérêts.

Prérequis

- Afin d'utiliser le SDK fournis par Google il faut utiliser une version LTS de Unity (2018 ou 2019).
- Obtenir une clé pour L'API Tile de Google*
- Avoir importé le SDK dans votre projet.
- Avoir au mois réaliser le premier chapitre "Chargement d'une zone du monde dans la scène"

*La facturation doit être activé sur le compte qui génère la clé d'API.

La base de donnée et l'API Par exemple admettons que l'on veuille placé une pancarte devant la gare de Neuchâtel stipulant justement que ce bâtiment est la gare de Neuchâtel. Pour ce faire ce point d'intérêt doit être stocké dans une base de donnée avec au minimum un nom et sa géolocalisation.

Dans notre cas, les points d'intérêts sont stockés dans une table de la base de donnée contenant les champs suivants :

- Nom
- Lat (Latitude)
- Lng (Longitude)

Une petite API permet de fournir ces informations au format JSON

Voici le code PHP de l'API

```php
<?php

$connexion = mysqli_connect("localhost", "root", "1234", "unityMaps");

//Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  exit();
}

$pointsOfInterestquery = "SELECT * FROM pointsOfInterest;";

$result = mysqli_query($connexion, $pointsOfInterestquery);

if ($result->num_rows > 0) {
  // output data of each row
  $points = [];
  while ($row = $result->fetch_assoc()) {
    $point->id = $row["id"];
    $point->name = $row["name"];
    $point->description = $row["description"];
    $point->lat = $row["lat"];
    $point->lng = $row["lng"];
    array_push($points,clone $point);
  }
  $pointsJSON = json_encode($points);
  echo $pointsJSON;
} else {
  echo "0 results";
}
$connexion->close();
```

Le code PHP n'est pas bien complexe, il ne fait que se connecté à une base de donnée, sélectionner toutes les entrées de la table contenant les points d'intérêts et les rendre au format JSON.

Du coté de Unity Maintenant que l'on a à notre disposition une API qui nous fournis les informations dont nous avons besoins au format JSON, il ne reste qu'à développer un composant capable d'aller chercher ces informations dans notre application Unity.

Pour se faire il faut commencer par créer un gameObject qui dans l'exemple se nomme `DatabaseController`.

Ensuite une classe C# est créée afin de représenter un objet point d'intérêts.

```
[System.Serializable]
public class PointOfInterest
{
    public float lat;
    public float lng;
    public string name;
    public string description;
}
```

Un Helper JSON est également créer afin de permettre la récupération des objets sous forme de tableau et vise vers cela

```
public static class JsonHelper
{
    public static T[] FromJson<T>(string json)
    {
        Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(json);
        return wrapper.Items;
    }

    public static string ToJson<T>(T[] array)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return JsonUtility.ToJson(wrapper);
    }

    public static string ToJson<T>(T[] array, bool prettyPrint)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return JsonUtility.ToJson(wrapper, prettyPrint);
    }

    [System.Serializable]
    private class Wrapper<T>
    {
        public T[] Items;
    }
}
```

Une fois ces outils créés, il reste à créer le composant utilisé par le GameObject `DatabaseController`, dans l'exemple il se nomme `Load Point Of Interest_04` 04 car il sera un peu différent dans le chapitre suivant.

Ce composant va aller chercher les informations fournis par l'API et les convertir directement en un tableau d'objets.

La fonction `Start()` de ce composant va lancer une co-routine c'est à dire une exécution non bloquante de la méthode spécifiée.

```
1  void Start() {
2      StartCoroutine(GetText());
3  }
```

La méthode `GetText()` va elle faire la lecture des informations fournies par l'API et la conversion du JSON en tableau d'objets `PointOfInterests`

```
1  IEnumerator GetText() {
2      UnityWebRequest www = UnityWebRequest.Get(url);
3      yield return www.SendWebRequest();
4
5      if(www.isNetworkError || www.isHttpError) {
6          Debug.Log(www.error);
7      }
8      else {
9          string jsonString = fixJson(www.downloadHandler.text);
10         this.pointsOfInterest = JsonHelper.FromJson<PointOfInterest>(
               jsonString);
11
12         AddGameObject();
13     }
14 }
```

La méthode `FixJson()` permet de formater le résultat

```
1  string FixJson(string value) {
2      value = "{\"Items\":" + value + "}";
3      return value;
4  }
```

La méthode `AddGameObject()` quand à elle va instancier des objets dans la scène Unity en convertissant les coordonnées de chaque point d'intérêts. Pour instancier un nouvel objet, il faut un modèle dans l'exemple le modèle est un GameObject rectangle contenant un texte.

```
1  public void AddGameObject(){
2      foreach(PointOfInterest point in this.pointsOfInterest){
3          LatLng ping = new LatLng(point.lat, point.lng);
4          Vector3 position = MapsService.Coords.FromLatLngToVector3(ping)
               ;
5          position.y = 2;
6
7          Quaternion rotation = new Quaternion(0, 0, 0, 1);
8          TextMeshPro txt = pointObject.transform.GetChild(0).
               GetComponent<TextMeshPro>();
9          txt.SetText(point.name);
10         objects.Add(Instantiate(pointObject,position,rotation));
11         objects[objects.Count-1].transform.SetParent(MapsService.
               transform,false);
12     }
13 }
```

A la fin, chaque objets maintenant contenus dans la liste voit son parents setter comme étant le gameObject MapsService afin que l'objet se déplace avec le monde.

L'exemple Pour tester cette fonctionnalité, vous trouverez une scène nommée `04_Load_objects_from_db` dans le projet d'exemple. Avant de lancer le programme, vous devez utilisé un serveur web afin de faire tourner l'API. Durant les tests la commande suivante a été utilisé dans le dossier où se trouve le fichier .php

```
1 php -S localhost:8000
```

et voici le résultat

Figure 7 – Résultat des points d'intérêts

5.4.6 Orienter le joueur

Ce chapitre décris l'étude d'une solution pour orienter le joueur dans le monde afin qu'il se dirige vers les points d'intérêts provenant de notre base de données.

Dans la scène de test vous retrouverez un contrôleur en vue à la troisième personne afin d'explorer la zone et vous rendre aux points d'intérêts marqués.

Prérequis

- Afin d'utiliser le SDK fournis par Google il faut utiliser une version LTS de Unity (2018 ou 2019).
- Obtenir une clé pour L'API Tile de Google*
- Avoir importé le SDK dans votre projet.
- Avoir réaliser le chapitre "Chargement d'objet provenant d'une base de donnée externe"

*La facturation doit être activé sur le compte qui génère la clé d'API.

La minimap Dans un premier temps l'idée d'utilisé une minimap semblait une bonne approche pour orienter le joueur. Un test a été réalisé en plaçant une grande sphère de couleur au dessus de chaque points d'intérêts et en utilisant une nouvelle caméra qui surplombe le joueur. Le système de layer de Unity permet de faire en sorte que certains GameObjects ne soient rendu que par certaines caméras. C'est cette solution qui était utilisé pour rendre les grandes sphères de couleur uniquement sur la caméra utilisée pour la minimap.

Mais une fois le test réalisé il s'avère qu'en fait ce n'est pas forcement la bonne approche. L'application finale du projet est destinée à être une application mobile et l'écran d'un mobile n'est pas suffisamment grand pour afficher une minimap tout en gardant une bonne vision sur le jeu. Surtout que l'application finale est pensée pour être utilisé en mode portrait comme Pokemon Go par exemple. Le compromis entre une minimap trop petite ou un affichage du jeu tronqué par cette dernière n'était donc pas la solution.

Les points de passage Les points de passage c'est quoi ? Ce sont des petits marqueurs orientés vers les points d'intérêts qui se bloquent sur les bords de l'écran. Cette idée semble plus juste pour un affichage sur smartphone. Toute fois elle a également ses limites car dans le cas où un nombre trop important de points d'intérêts sont affichés en même temps cela devient illisible.

Afin de mettre cette solution en place, il faut commencer par ajouter un canevas au projet. Puis ajouter un modèle de point de passage. Dans l'exemple, le point de passage modèle comprend également un texte qui permet d'afficher la distance entre le joueur et le point d'intérêt.

Il faut ensuite modifier le script `LoadPointsOfInterests` afin d'instancier un point de passage pour chaque point d'intérêt. Cela se fait dans la méthode `AddGameObject()`

```
listOfWayPoints.Add(Instantiate(waypoint_img) as Image);
listOfWayPoints[listOfWayPoints.Count-1].transform.SetParent(canvas.
    transform,false);
```

Maintenant il faut créer un nouveau script, dans l'exemple il se nomme `Waypoint` et ce composant sera attaché à la Caméra principal. Ce script aura pour but de placer les points de passage sur le canevas de tel sorte à ce qu'ils soient toujours dans la direction des points d'intérêts.

Tout se fait dans la méthode `Update()` du script

```
void Update() {
    for(int i = 0 ; i < targets.listOfWayPoints.Count; i++)
    {
        float minX = targets.listOfWayPoints[i].GetPixelAdjustedRect().
            width / 2;
        float maxX = Screen.width - minX;
        float minY = targets.listOfWayPoints[i].GetPixelAdjustedRect().
            height / 2;
        float maxY = Screen.height - minX;

        if (targets.objects.Count > 0 && targets.listOfWayPoints.Count
            > 0)
        {
            Vector2 pos = Camera.main.WorldToScreenPoint(offset +
                targets.objects[i].transform.position);

            if (Vector3.Dot((targets.objects[i].transform.position -
                transform.position), transform.forward) < 0)
            {
                //Target is behing the player
                if (pos.x < Screen.width / 2)
                {
                    pos.x = maxX;
                }
                else
                {
                    pos.x = minX;
                }
            }

            pos.x = Mathf.Clamp(pos.x, minX, maxX);
            pos.y = Mathf.Clamp(pos.y, minY, maxY);

            targets.listOfWayPoints[i].transform.position = pos;

            targets.listOfWayPoints[i].transform.GetChild(0).
                GetComponent<Text>().text = ((int)Vector3.Distance(
                targets.objects[i].transform.position, player.transform.
                position)).ToString() + "m";
        }
    }
}
```

5.4.7 Les textures et le Nine-Slicing

Ce chapitre décrit comment appliquer des textures sur les bâtiments et explique le principe du Nine-Slicing utilisé par Unity.

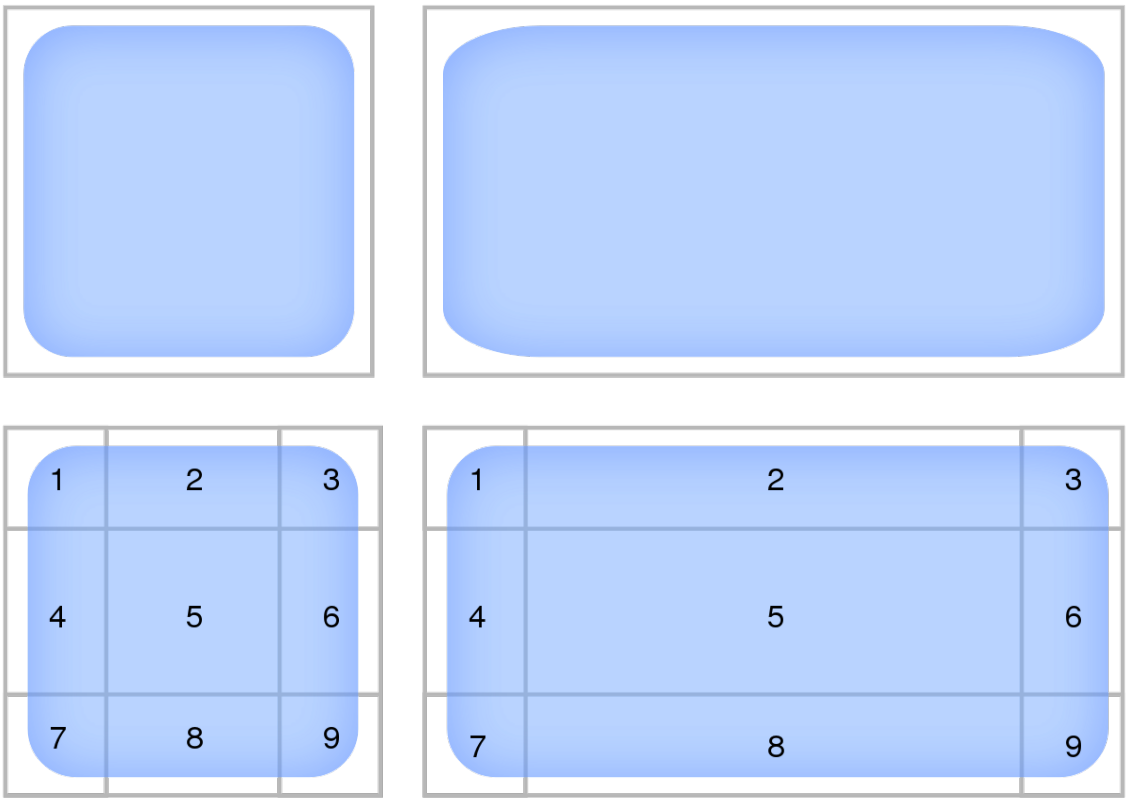
Dans la scène de test vous retrouverez un contrôleur en vu à la troisième personne afin d'explorer la zone et de voir l'application des textures sur les bâtiments.

Prérequis

- Afin d'utiliser le SDK fournis par Google il faut utiliser une version LTS de Unity (2018 ou 2019).
- Obtenir une clé pour L'API Tile de Google*
- Avoir importé le SDK dans votre projet.
- Avoir au mois réaliser le premier chapitre "Chargement d'une zone du monde dans la scène"

*La facturation doit être activé sur le compte qui génère la clé d'API.

Le Nine-silcing Le Nine-Slicing est une technique visant à découper les images en 9 carrés distincts. Si l'image doit être étirée alors seul les carrés du milieu seront étirés afin de conserver l'aspect des bords comme le montre l'image ci-dessous.



Dans notre cas, les carrés du milieu sont même répétés afin de dupliquer les zones de fenêtres des immeubles par exemple. Les images ci-après proviennent directement de la documentations du SDK

Les textures Afin d'appliquer une texture sur les bâtiments générés par le SDK il faut ajouter le composant `Nine Slicing` au GameObject `MapsSDK`, cela entraînera l'ajout automatique du composant `Building Texturer`. Ensuite il faut donner les matériaux correspondants aux textures des murs et des toits au composant `Building Texturer`. Dans l'exemple on en donne 6, le composant sélec-

tionnera aléatoirement entre les 6. Mais attention il doit y avoir le même nombre d'éléments dans les matériaux des murs que des toits.

5.4.8 L'API Playable Locations

En plus de l'API Semantic Tile permettant l'accès au SDK de Google Map dans unity, Google propose une deuxième API, Playable Locations API. Cette deuxième API permet l'accès à une base de données gigantesque contenant des informations sur une multitude de lieux.

Cette API sert des collections de points géographiques. Chaque points est choisi par Google en fonction de son aptitude à être utilisé dans un jeu basé sur des emplacements dans le monde réel.

Certains points sont situés à proximité de points d'intérêts importants, certains sont situés sur le trottoir, le long des routes, tandis que d'autres encore sont situés dans des parcs, terrains de jeu, places et d'autre zones accessible au public.

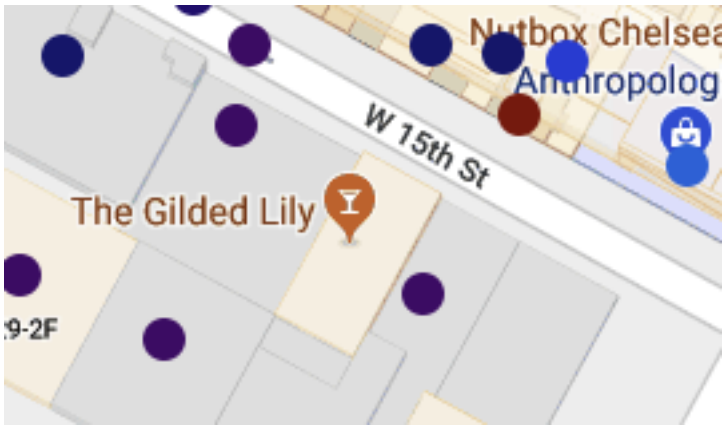


FIGURE 8 – Playable Locations

(Source : https://developers.google.com/maps/documentation/gaming/overview_locations)

Les données sont demandée avec le protocole HTTP à l'API et l'API renvoie ces dernières également en HTTP au format JSON.

La construction de la requête Pour contacter l'API le protocole HTTP sur le verbe POST est utilisé avec un corps de requête au format JSON.

Voici l'adresse : POST https://playablelocations.googleapis.com/v3:samplePlayableLocations?key=YOUR_API_KEY

Exemple de corps de requête :

```
{
  "areaFilter": {
    "s2CellId": string,
  },
  "criteria": [
    {
      "gameObjectType": number,
      "filter": {
          "maxLocationCount": number,
          "spacing": {
            "minSpacingMeters": number,
            "pointType": enum(PointType)
          },
          "includedTypes": [
            string
          ],
      },
      "fieldsToReturn": string
    }
  ]
}
```

L'objet `areaFilter` avec son paramètre `s2CellId` correspond à l'ID de la cellule S2 de la zone dans laquelle on veut obtenir des points.

Le tableau d'objet `criteria` comprend les différents critère par type d'objet.

Les différents type primaire :

TABLE 2 – Types primaire de l'API Playable Locations de Google

Type
education
entertainment
finance
food_and_drink
outdoor_recreation
retail
tourism
transit
transportation_infrastructure
wellness

Les autres paramètres sont assez parlant, `MaxLocationCount` = nombre maximal de point, `spacing` = l'espacement comprenant la distance minimal entre les points et une énumération des types de points impactés par l'espacement, etc.

Voici un schéma expliquant hiérarchiquement le corps d'une requête vers l'API

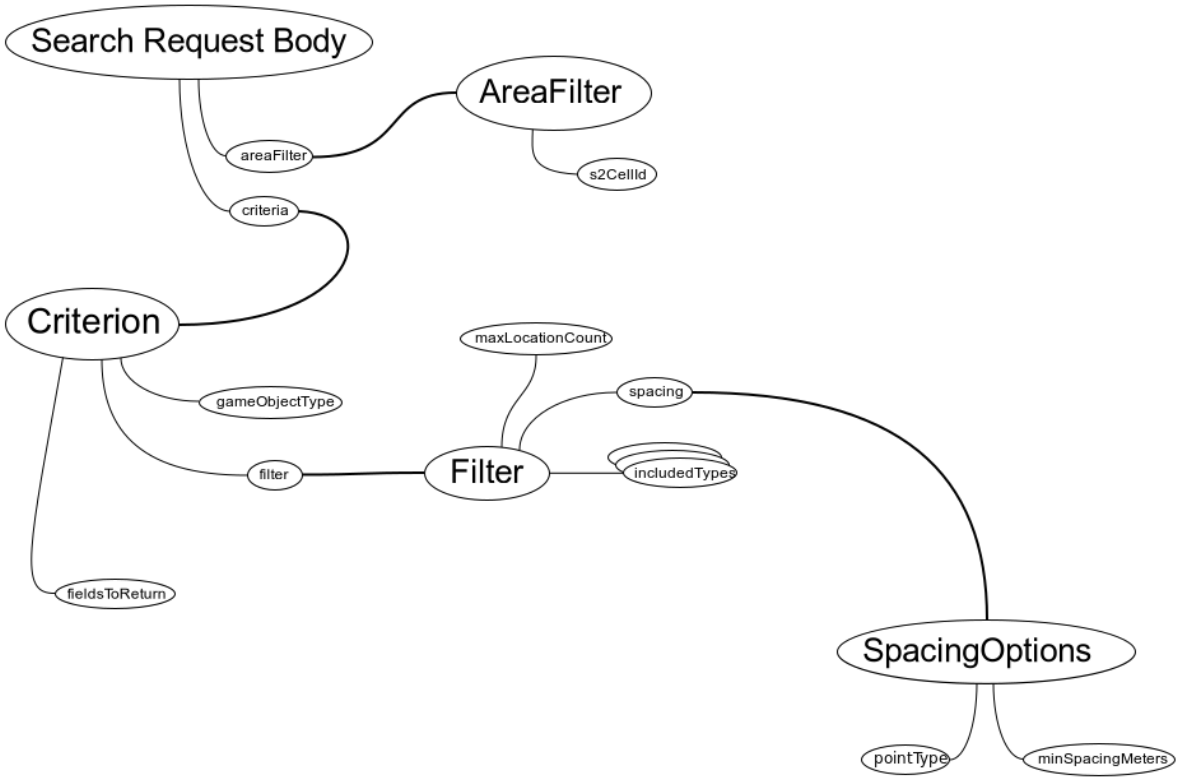


Figure 9 – Schema du corps d'une requête Playable Locations API

(Source : https://developers.google.com/maps/documentation/gaming/using_playable_locations)

La réponse de l'API En utilisant un outil comme Postman, il est possible de tester directement ce que l'API répond lorsque l'on envoie une requête.

Exemple de réponse :

```
{
    "locationsPerGameObjectType": {
        "1": {
            "locations": [
                {
                    "name": "curatedPlayableLocations/
                       ChIJdT4Bgj6uEmsRrhcUw_MU9MQ",
                    "placeId": "ChIJdT4Bgj6uEmsRrhcUw_MU9MQ",
                    "centerPoint": {
                        "latitude": -33.8693803,
                        "longitude": 151.19629889999996
                    }
                },
                {
                    "name": "curatedPlayableLocations/ChIJ1-
                       v38TauEmsRCk28fG54adI",
                    "placeId": "ChIJ1-v38TauEmsRCk28fG54adI",
                    "centerPoint": {
                        "latitude": -33.867492,
                        "longitude": 151.1942216
                    }
                }
            ]
        },
        "2": {}
    },
    "ttl": "86400s"
}
```

On s'aperçoit que les points rendus par l'API sont triés par type et contiennent un champs nom, ID et les coordonnées du centre du point sous forme de latitude et de longitude.

Ce qui nous intéresse le plus ici c'est le point avec ces coordonnées car par exemple dans Pokemon GO, ce que les points représentent importe peu, ce qui est intéressant c'est de placer des PokeStop(Endroit de récolte d'objets) à des endroits fixes et similaires pour tous les joueurs.

Les cellules S2 Dans le projet d'exemple vous ne trouverez pas de scènes d'exemples d'implémentation de ce chapitre car l'étudiant n'as pas eu le temps de réaliser cette dernière à cause des cellules S2.

Une cellules S2 c'est quoi ? Une cellule S2 c'est une découpe d'une région du monde plus ou moins grande. Chaque cellule à un S2cellID qui lui est propre et chaque cellule S2 est unique. La découpe du monde se fait pas la théorie des courbes d'Hiblert.

Pour en savoir plus : http://s2geometry.io/devguide/s2cell_hierarchy

Comment on obtient le S2CellID ? Il existe une librairie permettant de retourner le `S2CellID` mais le problème est là, la librairie n'est disponible qu'en Java, Python et C++. Malheureusement la librairie n'existe pas en C#. Toutefois la librairie a été testé dans un petit projet Python.

Pour obtenir un S2CellID il faut donner deux points sous forme de latitudes et longitudes formant une zone du monde.

Le test a été réalisé pour la ville de Neuchâtel mais malheureusement les S2CellID retournées n'étaient pas accepter par l'API Playable Locations.

De ce faite, le test n'est pas allé plus loin par manque de temps.

5.4.9 Les autres fonctionnalités

Le SDK de Google Maps est fournis avec une multitude d'exemple d'utilisation de fonctionnalités. La plupart des tests réalisés et décrits dans ce rapport se base sur ces exemples.

Les tests de ce rapport se sont focalisés sur les principales fonctionnalités utiles au développement de l'application finale du projet qui entend démontré un cas d'utilisation de ce SDK. Si vous désirez réaliser un projet demandant une fonctionnalités du SDK non décrites ni testez dans ce rapport n'hésitez pas à parcourir les exemples fournis par Google directement en important le SDK dans votre projet Unity. Les scripts C# des exemples sont bien commentés. De plus la documentation du SDK est peut être un peu maigre mais bien écrite et facilement accessible. (https://developers.google.com/maps/documentation/gaming/overview_musk)

5.5 Phase 3 : L'application Android

Cette partie du rapport décrit la dernière phase du projet. Cette phase conciste en le développement d'une applicaiton à destination d'un smart-phone Android. Cette application regrouppe une partie des fonctionnalités testées afin de permettre un exemple d'implémentationd du SDK. L'application se trouve dans le projet `02_Project/Guided_Tour_Neuchatel`-

Pour commencer l'application contient le même controlleur en vue à la troisième personne que les scènes d'exemple. Ensuite l'application contient également les fonctionnalités provenants de la deuxième phase suivante :

- Chargement de la zone en fonction du déplacement
- Collisions entre le joueur et les bâtiments
- Orientation du joueur avec way-points
- Insertion d'éléments provenant d'une base de données externes
- Le nineslicing pour l'application de texture sur les bâtiments

En ce qui concerne les points d'intérêts provenant de la base de données externes leur apparence a un peu évolué par rapport à la deuxième phase du projet. Il sont maintenant représenté que par une sphère bleue mais lorsque l'utilisateur clique dessus et maintient son clique, une modal affichant les informations tel que le nom et la déscription est affichée. De plus, lorsque l'utilisateur visite un point d'intérêt, il obtient 150 points de score en plus.

5.5.1 Le menu

Le menu est réalisé dans le canevas de la scène. Il est composé de différents éléments d'interface utilisateurs.

Le menu dans le jeu contient une page "A propos" contenant les informations obligatoires dans les projets de troisième année. C'est à dire les termes "Tout droits réservés", le logo de l'école et les noms des supérviseurs et étudiants ayant participé au projet.

Le menu contient également une page "Paramètre" contenant uniquement la possibilité de sélectionner la méthode de déplacement.

5.5.2 Les déplacements

Le joueur a la possibilité de séléctionner la méthode de déplacement qu'il désire dans le menu dans la page des paramètres. Il a le choix entre la méthode de déplacement avec joystick tactile et la méthode de déplacement en fonctions de la géolocalisation du smartphone.

Le Joystick La téchnique de déplacement avec le joystick permet de se déplacer dans la ville de Neuchâtel et partout ailleurs sans bouger physiquement.

Le joystick en lui même provient d'un paquet sur l'asset store. Il sagit de "Joystick pack". Plus précisement il sagit de l'objet `Fixed Joystick`

La méthode de déplacement est directement liée au composant et script `Player Movements`

Les positions du joystick sont récupérées de la même manière que n'importe quel input dans Unity

```
float horizontal = joystick.Horizontal;
float vertical = joystick.Vertical;
```

Pour le déplacement pur, l'angle de la camera est utilsé avec la direction définie par le joystick

```
Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;
Vector3 move = new Vector3(0f, 0f, 0f).normalized;

if (direction.magnitude >= 0.1f)
{

  float targetAngle = Mathf.Atan2(direction.x, direction.z) * Mathf.
     Rad2Deg + cam.eulerAngles.y;
  float angle = Mathf.SmoothDampAngle(transform.eulerAngles.y,
                                      targetAngle, ref
                                         turnSmoothVelocity,
                                         turnSmoothTime);
  transform.rotation = Quaternion.Euler(0f, angle, 0f);
  move = Quaternion.Euler(0f, targetAngle, 0f) * Vector3.forward;

}
```

Le déplacement est ensuite appliqué sur le monde et non le joueur en inversant la valeur en z

```
Vector3 tmp = move * speed * Time.deltaTime;
tmp.x = -tmp.x;
tmp.z = -tmp.z;
MapsService.transform.Translate(tmp);
```

La géolocalisation La méthode de suivi par géolocalisation du smartphone ne fonctionne que sur un smartphone. Cette méthode est directement liée au composant et au script `Player Movements Locations`

Pour récupérer la géolocalistion du téléphone le code suivant est utilisé

```
LatLng currentLocation =
  new LatLng(Input.location.lastData.latitude, Input.location.lastData.
     longitude);
Vector3 currentWorldLocation = MapsService.Coords.FromLatLngToVector3(
    currentLocation);
```

Ensuite le déplacement se fait toujours sur le monde en inversant la valeur de z

```
Vector3 targetCameraPosition = new Vector3(-currentWorldLocation.x,
                                           0, -currentWorldLocation.z);

MapsService.transform.position = Vector3.Lerp(MapsService.transform.
    position,
                                              targetCameraPosition,
                                                 Time.deltaTime * 5);
```

5.5.3 La génération et récolte de ressources

Des ressources, représentés par des cubes jaunes, sont générés aléatoirement autour du joueur au lancement de l'application. Puis à chaque fois que le joueur se déplace de cent mètres de nouvelles ressources sont également générés aléatoirement.

Elles sont au nombre de cinq et sont générés dans un pèrimètre de cent mètres autour du joueur.

Le script et composant permettant la génération des ressources est `Generate pieces` et se trouve dans l'objet `PieceGenerator`.

Le code permettant la génération avec `numberOfPiece` = 5

```
public void Spawn(){
  for(int i = 0 ; i < numberOfPiece ; i++){
    var position = new Vector3 (Random.Range(minPos,maxPos),1,Random.
       Range(minPos,maxPos));
    Quaternion rotation = new Quaternion(45, 45, 0, 1);
    GameObject piece = Instantiate(pieceModel,position,rotation);
    piece.transform.SetParent(MapsService.transform,false);
  }
}
```

Pour récolter les ressources, c'est dans le composant et script `Ressource Harvests` de l'objet `piece` que cela se passe.

```
if(isClicked){
  ScoringSystem.score += 50;
  Destroy(gameObject);
}
```

5.5.4 Le score

Le score est un système permettant de gameifier la visite. Il augmente comme vu précédement en visitant les points d'intérêts (150 points par points d'intérêts visité) et en récoltant les ressources au cours de la visite (50 points par ressources récoltée).

Le système de scoring se fait dans le script `ScoringController` et consiste simplement en la définition du variable static accessible depuis les autres scripts.

5.5.5 Le tactile du smartphone

L'application doit être déployée sur un smartphone android. Pour faire correspondre les entrées standards en entrées tactiles, il n'y a pas grand choses à faire. Unity gère assez bien le tactile. Cinemachine fonctionne très bien pour les mouvements de caméras avec le tactile.

Le problème c'est que cinemachine réagis à l'entier de l'écran et donc réagis lorsque l'utilisateur utilise le joystick. Pour contrer ce problème, le script `02_Project/Guided_Tour_Neuchatel/Library/PackageCache/com.unitycinemachine@2.3.4/Runtime/Core/CinemachineTouchInputMapper.cs` a été modifié de tel sorte à ne laisser à cinemachine que les 70% supérieur de l'écran.

```
1  public float TouchSensitivityX = 10f;
2  public float TouchSensitivityY = 10f;
3
4  public string TouchXInputMapTo = "Mouse X";
5  public string TouchYInputMapTo = "Mouse Y";
6
7  void Start()
8  {
9    CinemachineCore.GetInputAxis = GetInputAxis;
10 }
11
12 private float GetInputAxis(string axisName)
13 {
14   if (Input.touchCount > 0)
15   {
16     float min = Screen.height/100*30;
17     float max = Screen.height;
18
19     float inputY = Input.touches[Input.touchCount-1].position.y;
20
21     if(inputY >= min && inputY <= max){
22       if (axisName == TouchXInputMapTo)
23         return Input.touches[Input.touchCount-1].deltaPosition.x /
               TouchSensitivityX;
24       if (axisName == TouchYInputMapTo)
25         return Input.touches[Input.touchCount-1].deltaPosition.y /
               TouchSensitivityY;
26     }
27   }
28   return 0f;
29 }
```

6 Résultats

Les différentes phases du projet ont été réalisé avec succès. La première phase a permis d'avoir les notions de base nécessaire à la suite du projet. Cette phase a également permis de réaliser un contrôleur en vu à la troisième personne. La deuxième phase est la phase qui a pris le plus de temps, peut être même un peu trop, mais l'étude des différentes fonctionnalités proposées par le SDK de Google Maps permet maintenant à n'importe qui reprenant ce rapport et le projet d'exemple de mettre en place le SDK et d'implémenter les fonctions de base et nécessaires à n'importe quelle application utilisant ce dernier. Les tests ont été orienté vers l'application cible et donc la visite guidée d'une ville. La troisième phase est un succès également. L'application Android est un bon exemple de ce qu'il est possible de faire avec le SDK. Le design, l'UX et les textures ne sont pas forcément dès plus jolies ou intuitifs mais techniquement l'application est correcte.

Les objectifs principaux décrits dans le cahier des charges ont été entièrement réalisé. Les objectifs ont également tous été réalisé sauf un, sauvegarde local d'une visite. Mais en contrepartie, certaines choses non décrites dans le cahier des charges au départ ont été réalisées tel que l'étude de la deuxième API (Google API Playable Locations) permettant d'accéder à une base de données contenant des points d'intérêts dans le monde entier.

6.1 Points positifs

- La totalité des objectifs principaux décrit dans le cahier des charges ont pu être réalisé.
- La quasi-totalité des objectifs secondaires décrits dans le cahier des charges ont pu être réalisé.
- Application Android fonctionnelle et démonstrative.
- Documentation de tests complète accompagné de scènes d'exemples.

6.2 Points négatifs

- La forge aurait pu être utilisée de manière plus assidus.
- L'application n'as pas pu être réellement testée.

6.3 Améliorations

Dans un premier temps il pourrait être intéressant de revoir le système de collisions entre le joueur et les bâtiments dans le sens où collisionner avec ces derniers n'est peut être pas la meilleure solutions. Il pourrait être préférable de réduire l'opacité des bâtiments lorsque le joueur est à l'intérieur.

Il pourrait également être intéressant de scripter ou permettre l'application de filtres sur les points d'intérêts car lorsqu'il y en aura bien plus cela risque de devenir compliqué de les suivre tous en même temps.

Une idée de rentabilité du projet pour être l'intégration de sponsor au sein des points d'intérêts. Certains restaurateur, hôtelier ou encore commerçant pourraient payer pour obtenir l'ajout de leurs établissement comme étant un point d'intérêts dans le script de la visite d'une ville. Cette idée permettrait d'emmener les touristes vers les établissements des sponsor et ainsi leurs donner plus de visibilité. Il parait évident que dans cette idée seule la méthode de déplacement en fonction de la géolocalisation serait utilisée.

7 Tests et bugs

A ce jour, aucuns bug n'est connus. Mais cela n'entraine pas du tout que l'application est livrée sans bugs car peu de tests ont été réalisé.

La situation sanitaire ayant entrainé la dispense des cours à distance, l'étudiant ne se rendait plus du tout à neuchâtel. La partie de suivie de geolocalisation pour les déplacements du joueur ont été testé dans le village de Sonvilier. Malheureusement ou heureusement, le village de sonvilier ne contient pas autant de bâtiments que le centre ville de Neuchâtel. Les conditions n'étaient donc pas optimales pour réélement tester cette fonctionnalité.

La plannification initiale n'intégrait pas de temps dans le développement du projet pour réalisé une batterie de tests. Le temps est donc également un facteur clé dans le fait que malheureusement peu de tests ont été réalisé sur l'application.

8 Conclusion

En conclusion le projet s'est bien déroulé.

La phase de recherches et de tests était guidée par les différents exemples mis à disposition par le SDK. Six scènes de tests sont disponibles dans le projet `01_Tests/soft_example_tests`. Ces scènes sont directement liées à des chapitres de la partie réalisation de ce rapport. Dans l'ensemble les 6 scènes et chapitres permettent de comprendre comment le SDK fonctionne et donne des exemples d'implémentation de certaines problématiques nécessaire au développement de l'application finale.

La phase de développement de l'application Android s'est retrouvée un peu décalée par rapport à la planification car la phase de recherche a pris plus de temps que prévu. Mais ce n'était pas un réel soucis car beaucoup de fonctionnalités ont été reprises des tests pour l'implémentation de l'application.

Au final l'application remplis très bien sa fonction, avec la possibilité de visité différents points d'intérêts provenant d'une base de données externes, la récolte de ressources aléatoirement généré autours du joueur et la possibilité pour l'utilisateur de choisir entre un joystick tactile ou le suivi de la géolocalisation de son téléphone pour sa méthode de déplacement préférée.

Le cahier des charges est quasiment entièrement remplis. Les objectifs principaux ont tous été atteints et parmi les objectifs secondaires seul un n'est pas remplis. Il s'agit de la fonctionnalité de sauvegarde d'une visite en cours sur le téléphone de l'utilisateur. Mais en contrepartie, certaines choses n'étant pas présentes dans le cahier des charges ont été testées, documentées ou implémentées. Il y a, par exemple, l'étude de l'utilisation de la deuxième API proposé par Google (API Playable Locations) ou encore l'implémentation de Cinemachine pour le contrôleur en vu à la troisième personne.

Finalement, pour conclure ce rapport, il est important de souligner certains points. Premièrement, le SDK étant assez récent (sortie 2019, rapport écrit en 2020-21) il n'y a pas encore beaucoup de topics sur les forums ou ailleurs qui parlerait de problèmes, astuces, conseils, tuto ou autres sur le net. De manière général, seule la documentation fournis par Google et les script C# d'exemple sont présent pour nous aidez.

Deuxièmement, le SDK a un concurrent plus ancien que lui pour réaliser ce type de projet. Il se nome `Mapbox` celui-ci apparaît bien plus souvent dans les recherches internet sur les forums etc. Il peut être intéressant de bien se documenter sur les deux SDK (Mapbox et Google Maps) avant de se lancer dans un nouveau projet afin de définir celui qui conviendrait le mieux tout en gardant à l'esprit qu'une communauté est déjà présente autour de celui de Mapbox.

Pour finir, le SDK est tout de même puissant et assez simple à prendre en main de manière général. On obtient très rapidement quelque chose de correcte et si l'on a pas peur de creuser un peu dans la doc voir même directement dans les script C# d'exemple fournis avec le SDK, il est tout à fait possible d'en faire ce que l'on désire.

9 Annexes

Les fichiers suivant sont joints en annexes de ce rapport

- Plannification : `Gantt_of_project.xlsx`
- Cahier des charges : `CdC_DosSantosFerreiraJulien_Google Map dans Unity.doc`
- Journal de travail: `worklog.pdf`

10 Remerciements

Je remércie,

Senn Julien pour avoir suivis de manière assidu le projet.

Fridez Lucas pour avoir partagé son outil permettant de réaliser cette doccumentation en Markdown et d'obtenir un rapport au format PDF.

Océane Vogelbacher pour avoir relus et corrigé l'orthographe de ce rapport.

11 Références

[1] "Region Coverer," *Sidewalklabs.com*. [Online]. Available: https://s2.sidewalklabs.com/regioncoverer/. [Accessed: 28-Jan-2021].

[2] "Playable Locations API Overview," *Google.com*. [Online]. Available: https://developers.google.com/maps/documentation/gaming/overview_locations. [Accessed: 28-Jan-2021].

[3] "Maps SDK for unity overview," *Google.com*. [Online]. Available: https://developers.google.com/maps/documentation/gaming/overview_musk. [Accessed: 28-Jan-2021].

[4] "S2 Cells," *S2geometry.io*. [Online]. Available: http://s2geometry.io/devguide/s2cell_hierarchy. [Accessed: 28-Jan-2021].

[5] "Cinemachine Documentation," *Unity3d.com*. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/index.html. [Accessed: 28-Jan-2021].

[6] "THIRD PERSON MOVEMENT in unity," 24-May-2020. [Online]. Available: https://www.youtube.com/watch?v=4HpC–2iowE. [Accessed: 28-Jan-2021].

[7] "Waypoint Marker | Unity," 05-Mar-2019. [Online]. Available: https://www.youtube.com/watch?v=oBkfujKPZw8. [Accessed: 28-Jan-2021].

[8] "Maps, geocoding, and navigation APIs & SDKs," *Mapbox.com*. [Online]. Available: https://www.mapbox.com/. [Accessed: 28-Jan-2021].

[9] "Read JSON file data from server to unity c#," *Unity.com*. [Online]. Available: https://answers.unity.com/questions/935800/read-json-file-data-which-saved-in-server.html. [Accessed: 28-Jan-2021].

[10] "Unity using JSON from URL - EASY," 10-Apr-2019. [Online]. Available: https://www.youtube.com/watch?v=5j242VGGNZI. [Accessed: 28-Jan-2021].

[11] Unity Technologies, "Collision," *Unity3d.com*. [Online]. Available: https://docs.unity3d.com/ScriptReference/Collision.html. [Accessed: 28-Jan-2021].

[12] "Google Maps Platform," *Google.com*. [Online]. Available: https://developers.google.com/maps/documentation?hl=fr. [Accessed: 28-Jan-2021].

12 Signature

Sonvilier, le 28 janvier 2021

Dos Santos Ferreira Julien

Figure 10 – Signature de M.Dos Santos Ferreira